

Shell Scripting

What You Will Learn

- What scripts are.
- The components that make up a script.
- How to use variables in your scripts.
- How to perform tests and make decisions.
- How to accept command line arguments.
- How to accept input from a user.

Scripts

- Contain a series of commands.
- An interpreter executes commands in the script.
- Anything you can type at the command line, you can put in a script.
- Great for automating tasks.

script.sh

```
#!/bin/bash  
echo "Scripting is fun!"
```

```
$ chmod 755 script.sh  
$ ./script.sh  
Scripting is fun!  
$
```

Shebang

```
#!/bin/csh
```

```
echo "This script uses csh as the interpreter."
```

```
#!/bin/ksh
```

```
echo "This script uses ksh as the interpreter."
```

```
#!/bin/zsh
```

```
echo "This script uses zsh as the interpreter."
```

sleepy.sh

```
#!/bin/bash
```

```
sleep 90
```

```
$ ./sleepy.sh &
```

```
[1] 16796
```

```
$ ps -fp 16796
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
jason	16796	16725	0	22:50	pts/0	00:00:00	/bin/bash ./sleepy.sh

```
$
```

The interpreter executes the script

```
$ /tmp/sleepy.sh &
```

```
[1] 16804
```

```
$ ps -fp 16804
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
jason	16804	16725	0	22:51	pts/0	00:00:00	/bin/bash /tmp/sleepy.sh

```
$
```

```
$ ps -ef| grep 16804 | grep -v grep
jason    16804 16725    0 22:51 pts/0    00:00:00
/bin/bash /tmp/sleepy.sh
jason    16805 16804    0 22:51 pts/0    00:00:00
sleep 90
$ pstree -p 16804
sleepy.sh(16804)——sleep(16805)
$
```


Shebang or Not to Shebang

- If a script does not contain a shebang the commands are executed using your shell.
- You might get lucky. Maybe. Hopefully.
- Different shells have slightly varying syntax.

More than just shell scripts

```
#!/usr/bin/python  
print "This is a Python script."
```

```
$ chmod 755 hi.py  
$ ./hi.py  
This is a Python script.  
$
```

Variables

- Storage locations that have a name
- Name-value pairs
- Syntax:

```
VARIABLE_NAME="Value"
```

- Variables are case sensitive
- By convention variables are uppercase

Variable Usage

```
#!/bin/bash
```

```
MY_SHELL="bash"
```

```
echo "I like the $MY_SHELL shell."
```

```
#!/bin/bash
```

```
MY_SHELL="bash"
```

```
echo "I like the ${MY_SHELL} shell."
```

```
#!/bin/bash  
MY_SHELL="bash"  
echo "I am ${MY_SHELL}ing on my keyboard."
```

Output:

I am bashing on my keyboard.

```
#!/bin/bash
```

```
MY_SHELL="bash"
```

```
echo "I am $MY_SHELLing on my keyboard."
```

Output:

I am on mykeyboard.

Assign command output to a variable

```
#!/bin/bash
```

```
SERVER_NAME=$(hostname)
```

```
echo "You are running this script  
on ${SERVER_NAME}."
```

Output:

You are running this script on linuxserver

Assign command output to a variable

```
#!/bin/bash
```

```
SERVER_NAME=`hostname`
```

```
echo "You are running this script  
on ${SERVER_NAME}."
```

Output:

You are running this script on linuxserver

Variable Names

Valid:

```
FIRST3LETTERS="ABC"
```

```
FIRST_THREE_LETTERS="ABC"
```

```
firstThreeLetters="ABC"
```

Invalid:

```
3LETTERS="ABC"
```

```
first-three-letters="ABC"
```

```
first@Three@Letters="ABC"
```

Tests

Syntax:

```
[ condition-to-test-for ]
```

Example:

```
[ -e /etc/passwd ]
```

File operators (tests)

- d FILE True if file is a directory.
- e FILE True if file exists.
- f FILE True if file exists and is a regular file.
- r FILE True if file is readable by you.
- s FILE True if file exists and is not empty.
- w FILE True if the file is writable by you.
- x FILE True if the file is executable by you

String operators (tests)

-z STRING True if string is empty.

-n STRING True if string is not empty.

STRING1 = STRING2

True if the strings are equal.

STRING1 != STRING2

True if the strings are not equal

Arithmetic operators (tests)

arg1 -eq arg2 True if arg1 is equal to arg2.

arg1 -ne arg2 True if arg1 is not equal to arg2.

arg1 -lt arg2 True if arg1 is less than arg2.

arg1 -le arg2 True if arg1 is less than or equal to arg2.

arg1 -gt arg2 True if arg1 is greater than arg2.

arg1 -ge arg2 True if arg1 is greater than or equal to arg2

Making Decisions - The if statement

```
if [ condition-is-true ]  
then  
    command 1  
    command 2  
    command N  
fi
```

```
#!/bin/bash
```

```
MY_SHELL="bash"
```

```
if [ "$MY_SHELL" = "bash" ]
```

```
then
```

```
    echo "You seem to like the bash shell."
```

```
fi
```

Output:

You seem to like the bash shell.

if/else

```
if [ condition-is-true ]  
then  
    command N  
else  
    command N  
fi
```



```
#!/bin/bash
```

```
MY_SHELL="csh"
```

```
if [ "$MY_SHELL" = "bash" ]
```

```
then
```

```
    echo "You seem to like the bash shell."
```

```
else
```

```
    echo "You don't seem to like the bash  
shell."
```

```
fi
```

if/elif/else

```
if [ condition-is-true ]  
then  
    command N  
elif [ condition-is-true ]  
then  
    command N  
else  
    command N  
fi
```

```
#!/bin/bash
```

```
MY_SHELL="csh"
```

```
if [ "$MY_SHELL" = "bash" ]
```

```
then
```

```
    echo "You seem to like the bash shell."
```

```
elif [ "$MY_SHELL" = "csh" ]
```

```
then
```

```
    echo "You seem to like the csh shell."
```

```
else
```

```
    echo "You don't seem to like the bash or csh shells."
```

```
fi
```

For loop

```
for VARIABLE_NAME in ITEM_1 ITEM_N
do
    command 1
    command 2
    command N
done
```

```
#!/bin/bash
for COLOR in red green blue
do
    echo "COLOR: $COLOR"
done
```

Output:

COLOR: red

COLOR: green

COLOR: blue

```
#!/bin/bash
COLORS="red green blue"

for COLOR in $COLORS
do
    echo "COLOR: $COLOR"
done
```

```
#!/bin/bash
PICTURES=$(ls *jpg)
DATE=$(date +%F)

for PICTURE in $PICTURES
do
    echo "Renaming ${PICTURE} to ${DATE}
-${PICTURE}"
    mv ${PICTURE} ${DATE}-${PICTURE}
done
```

```
$ ls
```

```
bear.jpg  man.jpg  pig.jpg  rename-pics.sh
```

```
$ ./rename-pics.sh
```

```
Renaming bear.jpg to 2015-03-06-bear.jpg
```

```
Renaming man.jpg to 2015-03-06-man.jpg
```

```
Renaming pig.jpg to 2015-03-06-pig.jpg
```

```
$ ls
```

```
2015-03-06-bear.jpg  2015-03-06-man.jpg
```

```
2015-03-06-pig.jpg  rename-pics.sh
```

```
$
```


Positional Parameters

```
$ script.sh parameter1 parameter2 parameter3
```

```
$0 : "script.sh"
```

```
$1 : "parameter1"
```

```
$2 : "parameter2"
```

```
$3 : "parameter3"
```

```
#!/bin/bash
```

```
echo "Executing script: $0"
```

```
echo "Archiving user: $1"
```

```
# Lock the account
```

```
passwd -l $1
```

```
# Create an archive of the home directory.
```

```
tar cf /archives/${1}.tar.gz /home/${1}
```

```
$ ./archive_user.sh elvis
```

```
Executing script: ./archive_user.sh
```

```
Archiving user: elvis
```

```
passwd: password expiry information changed.
```

```
tar: Removing leading `/' from member names
```

```
$
```

```
#!/bin/bash
```

```
USER=$1 # The first parameter is the user.
```

```
echo "Executing script: $0"
```

```
echo "Archiving user: $USER"
```

```
# Lock the account
```

```
passwd -l $USER
```

```
# Create an archive of the home directory.
```

```
tar cf /archives/${USER}.tar.gz /home/${USER}
```

```
#!/bin/bash
```

```
echo "Executing script: $0"
```

```
for USER in $@
```

```
do
```

```
    echo "Archiving user: $USER"
```

```
    # Lock the account
```

```
    passwd -l $USER
```

```
    # Create an archive of the home directory.
```

```
    tar cf /archives/${USER}.tar.gz /home/${USER}
```

```
done
```

```
$ ./archive_user.sh chet joe
Executing script: ./archive_user.sh
Archiving user: chet
passwd: password expiry information changed.
tar: Removing leading `/' from member names
Archiving user: joe
passwd: password expiry information changed.
tar: Removing leading `/' from member names
$
```

Accepting User Input(STDIN)

The read command accepts STDIN.

Syntax:

```
read -p "PROMPT" VARIABLE
```

```
#!/bin/bash
```

```
read -p "Enter a user name: " USER  
echo "Archiving user: $USER"
```

```
# Lock the account  
passwd -l $USER
```

```
# Create an archive of the home directory.  
tar cf /archives/${USER}.tar.gz /home/${USER}
```



```
$ ./archive_user.sh
```

```
Enter a user name: mitch
```

```
Archiving user: mitch
```

```
passwd: password expiry information changed.
```

```
tar: Removing leading `/' from member names
```

```
$
```

Summary

```
#!/path/to/interpreter
```

```
VARIABLE_NAME="Value"
```

```
$VARIABLE_NAME
```

```
${VARIABLE_NAME}
```

```
VARIABLE_NAME=$(command)
```

```
if [ condition-is-true ]  
then  
    commands  
elif [ condition-is-true ]  
then  
    commands  
else  
    commands  
fi
```

For Loop

```
for VARIABLE_NAME in ITEM_1 ITEM_N  
do  
    command 1  
    command 2  
    command N  
done
```

Summary, continued.

Positional Parameters:

`$0, $1, $2 ... $9`

`$@`

Comments start with **#**.

Use **read** to accept input.